**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторной работе №5**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: Бинарные деревья поиска**

Студент гр. 8304 _____ Ивченко А.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

**Цель работы.**

Ознакомиться со структурой бинарных деревьев поиска; изучить вариант обхода бинарного дерева от меньшего элемента к большему

**Задание(вариант 12).**

1)По заданному файлу F (типа file of Elem), все элементы которого различны, построить
структуру данных определённого типа – БДП или хеш-таблицу;
2в) Записать в файл элементы построенного БДП в порядке их возрастания; вывести построенное БДП на экран.

**Описание работы алгоритма.**

Алгоритм работы программы совмещен с функцией высвобождения памяти, т. к. обход бинарного дерева поиска для удаления элементов совпадает с обходом дерева от меньшего элемента к большему: сначала идет вызов левого поддерева пока не найдется нулевое поддерево, затем записывается корень этого дерева и правое поддерево (тоже пока не найдется нулевое поддерево) . Для реализации этого алгоритма были реализованы методы классической вставки а также вставки с рандомизацией.

Была реализована структура Node, имеющая поля: ключ, высота БДП, указатели на поддеревья, которые по умолчанию равны нулю, также есть явный метод, устанавливающий значение корня.

```
struct node
{
	int key;
	int size;
	node* left;
	node* right;
	node(int k){
		key = k;
		left = right = 0;
		size = 1;
	}
```

};

**Краткое описание функций.**

**node* find(node*, int) — поиск ключа в дереве**

**node* insert(node* p, int k) — классическая вставка**

**int getsize(node* ) - возвращает высоту**

**void fixsize(node* ) - корректировка высоты**

**node* rotateright(node* ); node* rotateleft(node* ) - правый и левый поворот**

**node* insertroot(node*, int); - вставка с рандомизацией**

**node* join(node*, node*) - объединение поддерьев**

**node* remove(node*, int) — удаление элемента**

**Тестирование.**

```
Изображение бинарного дерева поиска из заданного набора элементов:12 32 432 1
 12 32 432
   1

Перечисление элементов в порядке возрастания:1 12 32 432


Изображение бинарного дерева поиска из заданного набора элементов:32 2 3 5345 342
 342 5345
   32
     3
       2

Перечисление элементов в порядке возрастания:2 3 32 342 5345


Изображение бинарного дерева поиска из заданного набора элементов:9 12 320 123
 9 12 320
       123

Перечисление элементов в порядке возрастания:9 12 123 320


Некорректные данные:fsdf afsd
```

**Вывод.**

В результате лабораторной работы был получен опыт по работе с бинарными деревьями поиска. Были реализованы метода, необходимые для выполнения задачи.

# ИСХОДНЫЙ КОД

```
#INCLUDE <IOSTREAM>
#INCLUDE <FSTREAM>
#INCLUDE <SSTREAM>
#INCLUDE <CSTDLIB>
#INCLUDE <VECTOR>
#INCLUDE <CCTYPE>

USING NAMESPACE STD;

STRUCT NODE
{
	INT KEY;
	INT SIZE;
	NODE* LEFT;
	NODE* RIGHT;
	NODE(INT K){
		KEY = K;
		LEFT = RIGHT = 0;
		SIZE = 1;
	}
};

NODE* FIND(NODE*, INT);
NODE* INSERT(NODE*, INT);
INT GETSIZE(NODE* );
VOID FIXSIZE(NODE* );
NODE* ROTATERIGHT(NODE* );
NODE* ROTATELEFT(NODE* );
NODE* INSERTROOT(NODE*, INT);
NODE* JOIN(NODE*, NODE*);
NODE* REMOVE(NODE*, INT);
VOID PRINTANDDESTROY(NODE* &B, STD::OFSTREAM &FOUT);

TYPEDEF NODE *BINSEARCHTREE;
```

```
NODE* FIND(NODE* P, INT K)
{
   IF( !P ) RETURN 0;
   IF( K == P->KEY )
      RETURN P;
   IF( K < P->KEY )
      RETURN FIND(P->LEFT,K);
   ELSE
      RETURN FIND(P->RIGHT,K);
}

INT GETSIZE(NODE* P)
{
        IF( !P ) RETURN 0;
        RETURN P->SIZE;
}

VOID FIXSIZE(NODE* P)
{
        P->SIZE = GETSIZE(P->LEFT)+GETSIZE(P->RIGHT)+1;
}

NODE* ROTATERIGHT(NODE* P)
{
        NODE* Q = P->LEFT;
        IF( !Q ) RETURN P;
        P->LEFT = Q->RIGHT;
        Q->RIGHT = P;
        Q->SIZE = P->SIZE;
        FIXSIZE(P);
        RETURN Q;
}

NODE* ROTATELEFT(NODE* Q)
{
        NODE* P = Q->RIGHT;
        IF( !P ) RETURN Q;
        Q->RIGHT = P->LEFT;
        P->LEFT = Q;
        P->SIZE = Q->SIZE;
```

```
        FIXSIZE(Q);
        RETURN P;
}

NODE* INSERTROOT(NODE* P, INT K)
{
        IF( !P ) RETURN NEW NODE(K);
        IF( K<P->KEY )
        {
                P->LEFT = INSERTROOT(P->LEFT,K);
                RETURN ROTATERIGHT(P);
        }
        ELSE
        {
                P->RIGHT = INSERTROOT(P->RIGHT,K);
                RETURN ROTATELEFT(P);
        }
}

NODE* INSERT(NODE* P, INT K)
{
        IF( !P ) RETURN NEW NODE(K);
        IF( RAND()%(P->SIZE+1) == 0 )
                RETURN INSERTROOT(P,K);
        IF( P->KEY>K )
                P->LEFT = INSERT(P->LEFT,K);
        ELSE
                P->RIGHT = INSERT(P->RIGHT,K);
   FIXSIZE(P);
        RETURN P;
}

NODE* JOIN(NODE* P, NODE* Q)
{
        IF( !P ) RETURN Q;
        IF( !Q ) RETURN P;
        IF( RAND()%(P->SIZE+Q->SIZE) < P->SIZE )
        {
                P->RIGHT = JOIN(P->RIGHT,Q);
                FIXSIZE(P);
```

```
            RETURN P;
        }
        ELSE
        {
            Q->LEFT = JOIN(P,Q->LEFT);
            FIXSIZE(Q);
            RETURN Q;
        }
}

NODE* REMOVE(NODE* P, INT K)
{
        IF( !P ) RETURN P;
        IF( P->KEY==K )
        {
            NODE* Q = JOIN(P->LEFT,P->RIGHT);
            DELETE P;
            RETURN Q;
        }
        ELSE IF( K<P->KEY )
            P->LEFT = REMOVE(P->LEFT,K);
        ELSE
            P->RIGHT = REMOVE(P->RIGHT,K);
        RETURN P;
}

VOID DISPLAYBT(NODE* B, INT N, STD::OFSTREAM &FOUT){
            IF (B != NULLPTR) {
                FOUT << ' ' << B->KEY;
                IF(B->RIGHT != NULLPTR) {
                    DISPLAYBT(B->RIGHT, N + 1, FOUT);
                }
                ELSE FOUT << ENDL;
                IF(B->LEFT != NULLPTR){
                    FOR (INT I = 1; I <= N; I++)
                        FOUT << "  ";
                    DISPLAYBT(B->LEFT, N + 1, FOUT);
                }
            }
}
```

```
VOID PRINTANDDESTROY(NODE* &B, STD::OFSTREAM & FOUT)
{
	IF (B != NULLPTR) {
		PRINTANDDESTROY(B->LEFT , FOUT);
		FOUT << B->KEY << ' ';
		PRINTANDDESTROY(B->RIGHT, FOUT);
		DELETE B;
		B = NULLPTR;
	}

}
INT CHECK(STD::STRINGSTREAM &SS, STD::STRING S){
	CHAR CHECK;
	WHILE (SS >> CHECK){
			IF ((CHECK != ' ') && (!ISDIGIT(CHECK)))
					RETURN 0;
	}
	RETURN 1;
}

VOID READFROMFILE(STD::IFSTREAM &FILE){



		STD::STRINGSTREAM SS;
		STD::STRINGSTREAM CHECKS;
		VECTOR<INT> ARR;
		IF(!FILE.IS_OPEN()){
			STD::COUT<<"НЕВЕРНЫЙ ПУТЬ К ФАЙЛУ\N";
			RETURN;
		}

		STD::OFSTREAM FOUT;
		FOUT.OPEN ("TESTS/OUTPUT.TXT", STD::IOS::APP);
		IF(!FOUT.IS_OPEN()){RETURN;}

		STD::STRING S;
		WHILE (STD::GETLINE(FILE,S)){

			SS << S;
```

```
                CHECKS << S;
                IF(CHECK(CHECKS, S)){



                        INT CUR;
                        WHILE (SS >> CUR)
                                ARR.PUSH_BACK(CUR);



                        BINSEARCHTREE MYBINTREE = NULLPTR;
                        FOR (AUTO I: ARR) {
                                MYBINTREE = INSERT(MYBINTREE, I);
                        }

                        FOUT << "ИЗОБРАЖЕНИЕ БИНАРНОГО ДЕРЕВА ПОИСКА ИЗ
ЗАДАННОГО НАБОРА ЭЛЕМЕНТОВ:" << S << ENDL;

                        DISPLAYBT (MYBINTREE,1, FOUT);

                        FOUT << "\NПЕРЕЧИСЛЕНИЕ ЭЛЕМЕНТОВ В ПОРЯДКЕ
ВОЗРАСТАНИЯ:";



                        ARR.CLEAR();
                        S.CLEAR();
                        PRINTANDDESTROY(MYBINTREE, FOUT);
                        FOUT << "\N\N\N";
                }ELSE FOUT << "НЕКОРРЕКТНЫЕ ДАННЫЕ:" << S << ENDL;
                SS.CLEAR();
                CHECKS.CLEAR();
        }


}
```

```
INT MAIN(INT ARGC, CHAR* ARGV[]){


        IF (ARGC == 2) {
                STD::IFSTREAM FILE(ARGV[1]);
                READFROMFILE(FILE);
        }



        RETURN 0;
}
```